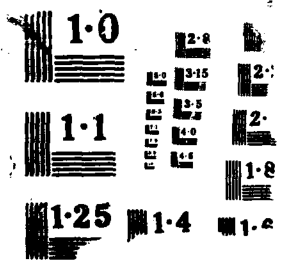
AD-A195 600 SIMONE A DISCRETE EVENT SIMULATION SUPERVISOR(U) 1/1
DEPARTMENT OF DEFENCE CANBERRA (AUSTRALIA) CENTRAL
STUDIES BRANCH B K MCMILLAN MAR 88 CSB-MEMO-2
UNCLASSIFIED DODA-AR-005-255 F/G 12/5 NL


1·0
1·1
1·25
1·4
2·8
3·15
3·5
4·0
4·5
1·8

CENTRAL STUDIES BRANCH
Canberra A.C.T.

AD-A195 600

Department of Defence

COPY NO. 11 OF 55

UNCLASSIFIED

CSB MEMORANDUM 2

MARCH 1988

AR-005-255

SIMONE, A DISCRETE EVENT SIMULATION SUPERVISOR

BY B.K. McMILLAN

UNCLASSIFIED

CSB MEMORANDUM 2

JANUARY 1988

SIMONE - A DISCRETE EVENT SIMULATION SUPERVISOR

B.K. McMILLAN

ABSTRACT

The general discrete event simulation supervisory system SIMONE is described. The facilities available and method of use are detailed and illustrated with an example. It uses a three phase system to control time and event selection, and provides additional mathematical, random sampling and data collection facilities. Systems with a PASCAL compiler should be able to use SIMONE.

TABLE OF CONTENTS

LIST OF TABLES

TABLE OF CONTENTS (Cont'd)

Page No.

LIST OF FIGURES

INTRODUCTION

1. In the early 1960s, P.R. Hills of Bristol College of Science and Technology generated the approach and necessary programming for SIMON, a supervisory system and framework for discrete event simulation (References 1 to 3). The approach requires a model description based on three separate aspects or phases (A, B and C), described later. To be effective, it requires a supervisory system to handle lists, times and event selection. The original supervisor was programmed in ALGOL as a set of subroutines, collectively called SIMON. Practical evaluation and expansion of the system was carried out in conjunction with a UK steel company, Richard Thomas and Baldwins, primarily in their Operational Research Department.

2. Recently, while using a PASCAL version of SIMON for a large simulation, the author found certain drawbacks and overcome them by extending the system. More recently, the original conversion to PASCAL was improved by using pointers for all lists and further extended. The extensions include a generalisation and enhanced flexibility of entity and event handling as well as a removal of restrictions on histogram and interpolation table sizes. This extended system was named SIMONE.

3. Advantages of the SIMONE system include:

   a. its portability (most computers of all sizes have PASCAL available);

   b. its flexibility to handle most simulation tasks, particularly discrete event models;

   c. the associated advantages of structured programming that its framework encourages; and

   d. the ease with which it can be learned and used.

Keywords : Simulation Languages ; Monte Carlo method ;

4. Thus for example models can be developed on a mainframe with extensive debugging facilities and good compilation facilities and then transferred to, say, a PC for regular use. Experience has shown that its flexibility and structured approach allow very significant savings in construction and debugging time as well as improving the reliability and capacity to modify models after initial use. Often, Operational Research workers and analysts find that a particular tool is used for a short period and then is not needed for some considerable time. Under these conditions SIMONE is an excellent choice because once understood it is very easy to re-learn and apply.

Australia (CR)

5. As documentation for SIMON is not readily available in Australia, the whole SIMONE system and approach will be described in this Memorandum.

6. When modelling a problem it is necessary to make decisions such as form of information to be extracted from the model and information available (i.e. the scope of the model) as well as time available to do the job. These aspects are dealt with in many texts and will not be covered here.

7. This paper assumes that the reader has a rudimentary knowledge of PASCAL, and of the aims and limitations of simulation.

8. An example of cars being served is included to show the general form of a SIMONE based simulation. It is recommended that the example be studied in detail before using SIMONE.

BASIC CONCEPTS

Discrete and Continuous Modelling

9. Simulation problems are often classified by the dominant characteristic of parameter variation. Where parameters are primarily influenced by specific events as might occur in product manufacture through a series of processes, then discrete event simulation should be appropriate. Where parameters vary steadily with time as might occur in water flows through hydro schemes, biological modelling or in macro economic areas, then continuous simulation modelling should be appropriate. Generally speaking, both types of problem can be handled with either type of approach, but perhaps with a little more difficulty. In large and complex problems this extra difficulty is likely to be detrimental to speed of construction, general model comprehension, ease of debugging, reliability and extendability. SIMONE has discrete event orientation, and therefore does not include continuous simulation language facilities such as integration and differential equation solving unless these are available through the normal computer library functions.

Describing The Model

10. Commonly, flow diagrams are drawn to describe the logic and information flows and structures abstracted from the 'Real World' situation. Without a suitable thinking framework such diagrams can rapidly get out of control - in size, complexity, readability, and consequently reliability.

11. SIMONE provides a formal or generalised flow diagram (Figure 1) which fits all relevant real world circumstances, and which encourages a structured approach to the problem. Control of time is in phase A, while the modelling is in the phases B and C.

12. The real world problem being modelled must be conceptually separated into a number of substantially independent time dependant processes that will be called phase B events. These events are linked in two ways: firstly by the entities being processed (i.e, having the event happen to them) and secondly, by other linking processes (phase C events), which happen as a result of phase B events. The whole is co-ordinated by phase A which advances time and indicates the next event. A very simple example might be a train set, where a train is an entity, the timetable is phase A, stops are phase B events, and a signal for entry to a section of line is a phase C event. There is only a small requirement for special terminology in contrast with some other systems.

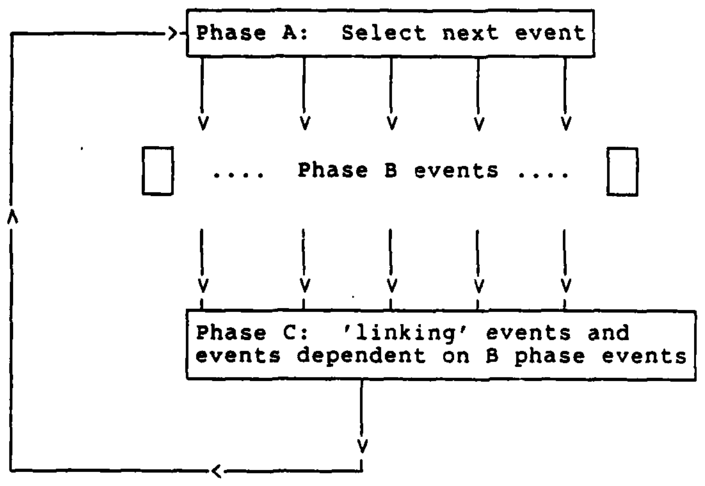

Phase A: Select next event
.... Phase B events ....
Phase C: 'linking' events and events dependent on B phase events

Figure 1. GENERAL SIMONE BASED STRUCTURE

13. Such a formal representation is the kernel of the problem. Simulations frequently repeat runs of the kernel with variations in parameters. Thus the final program will include looping controls and suitable initialisation components.

14. Tackling problems in this way allows the program to be highly structured, and permits construction of stand-alone modules. Other forms of flow diagram are still useful within this framework. For example an object arriving for a service could be as in Figure 2.

15. Relationships between entities and events can also be portrayed using another form of diagram in which each entity is represented by a circle and events of interest are placed on the circle. The example is illustrated using this method in Figure 3. Note that a number of events can be collapsed into a single phase B event if convenient.

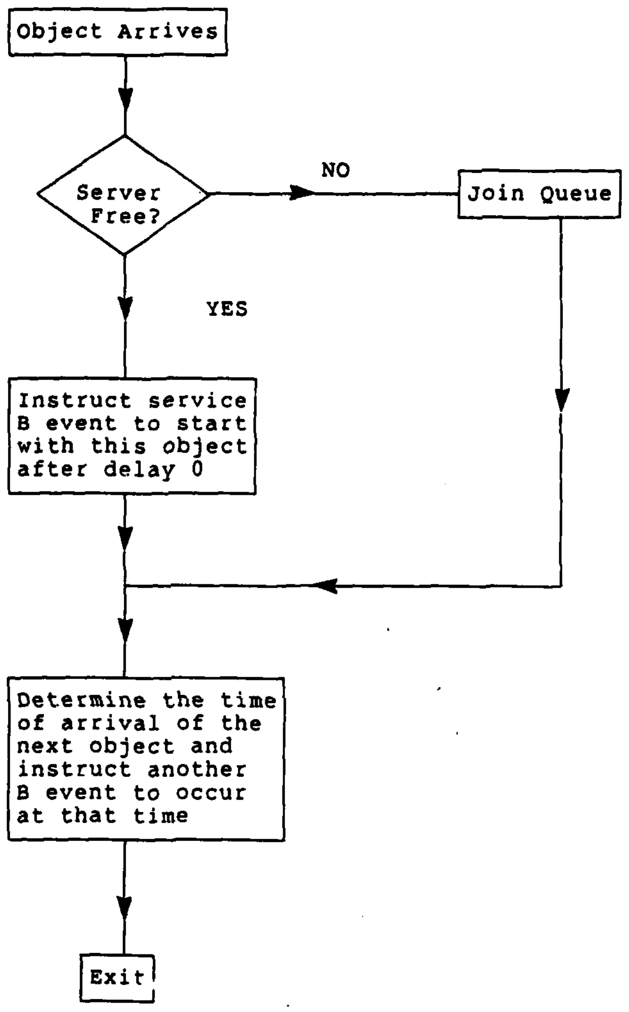

Object Arrives
Server Free?
NO
Join Queue
YES
Instruct service B event to start with this object after delay 0
Determine the time of arrival of the next object and instruct another B event to occur at that time
Exit

Figure 2. A PHASE B EVENT FLOW DIAGRAM

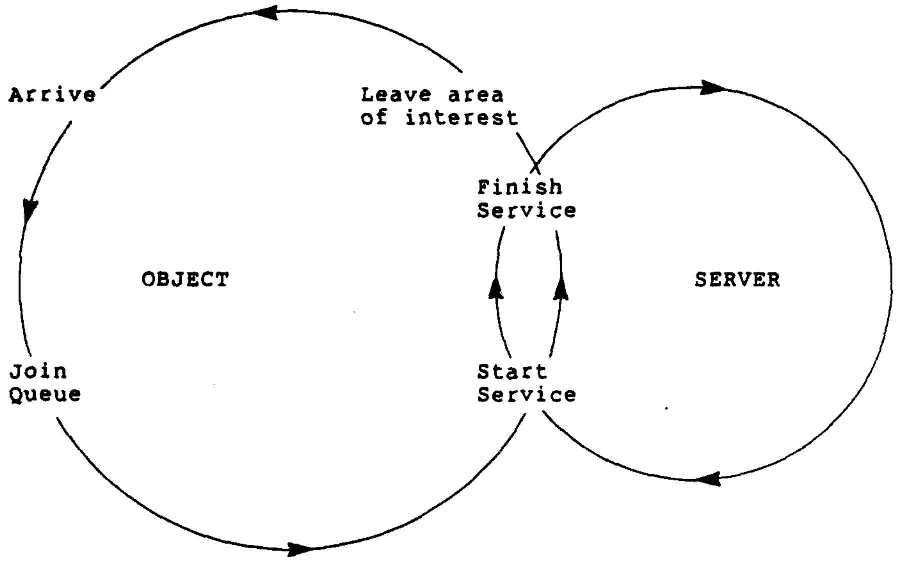


Figure 3. OBJECT-SERVER INTERACTION

Entities, Events and Time

16. An entity will normally be an object, person or conceptually solid 'thing'. Events will normally be actions, processes or occurrences, each with an associated time of occurrence. Entities have attributes and are processed through events which may influence those attributes. Typically an entity could be a customer in a shop, an item being manufactured, an aircraft in flight, or a radar. An event could be the arrival of a customer at a shop, an aircraft taking off, a radar scanning an area.

17. To make use of such entities and events, time and time handling are needed. With each entity there is associated a 'next' event and a time at which that event will occur. Control of the sequence of event and entity processing (in phase A) is through, a list of forthcoming events, each event having an associated time and entity. The list is scanned for the next event to occur and program control transferred to that event after the list entry has been removed.

18. As indicated earlier, events can be in one of two classes. Events in the first class are gathered together in Phase B and occur at times that have been pre-determined. Events in the second class are gathered together in Phase C and only occur as a results of a set of events or conditions rather than at a time. Events which follow at some time after an event or condition would generally be classified as Phase B events.

Continuous Elements

19. Continuously varying parameters can be handled by one or more of three methods:

a. introduce Phase B events to update values on a regular or convenient basis;

b. introduce Phase C events to update as events dictate; or

c. add components to Phase A which update values to 'current' simulation time; this enables the use of these values in events occurring at that time.

A typical application of Method c is to advance all travelling objects. Adjustments to rates and directions of travel can be made using events.

Entity Attributes

20. Objects in a simulation need to have properties that can be examined and altered. Their definition is left to the programmer, but there must be a link to the SIMONE supervisory system. This is provided by an additional attribute (of type entity_ptr) that must be included with the type definition for each entity. For example, some 'car' entities may be needed, each one having an index number and a time of joining a queue. A suitable PASCAL type definition might be:

```
type
  cartype=record
    e:entity_ptr;
    join_q_time:real;
  end;
```

When the variable 'cars' is defined, it may be declared as an array of cartype, the array index being the car index number:

```
var
  car :array[1..10] of cartype;
```

21. The link to the supervisory system gives the entity some 'housekeeping' properties that allow identification of an event and a time. These attributes are available through the 'entity_ptr' element and will be discussed later, but as an example, the time of the next event to happen to car i would be accessed through:

```
  car[i].e^.time
```

Composite Entities

22. In the original SIMON, an entity was essentially a unique object. Several objects with the same properties (excepting an index number) were defined as group-entities. Originally it was expected that each entity should only relate to a single event, but early versions of SIMONE introduced the multiple group-entity (MGE) which relates to several events. In the current version, apart from one procedure, there is no specific coding for the MGE or group entity, since an entity may be made to relate to any event, and becomes a group entity simply by defining the relevant variable to be an array. The concept of the group entity is still used in this paper however, and the MGE implemented by a requirement to define the relevant event for an entity when it is added to the list of forthcoming events.

DETAILS OF SIMONE

23. Having decided to adopt the discrete event three phase approach to a simulation, the reader will need exact details of SIMONE. The coding itself (Reference 7) is a good source of information but since it is nearly 900 lines, a summary of the salient points and some explanations may be needed. In addition there is an example with discussion in the next section that may help the novice user, particularly if this section is initially just skimmed. A Glossary has also been included for the novice user.

Essential Entity Attributes

24. These attributes are defined in the SIMONE module in the types entity_ptr (simply a pointer to entity) and entity:

```
entity_ptr=^entity;
entity=record
  refnum,memnum,typenum   :integer;
  time                    :real;
end;
```

Elements refnum and time give access to the event and its time of occurrence, while memnum is the index number of the entity. In the cars example above, memnum could be set to the array index value. Element typenum has been included for cases where more than one type of entity can enter an event, although such situations should be avoided where possible. The user can always expand the definition of entity if required for a particular application.

25. Some of these attributes need to be defined in the initialisation section if group entities are used. In addition it is often convenient to hold a list of those that are not being actively used by the simulation. For this purpose the procedure ENTITY_ELEMENT should be used, in which MEMNUM and TYPENUM are defined, and the element is added to the end of a list. Using the car example of the last section, coding could be:

```
  declarelist(free_cars,'free cars');
  for i:=1 to 10 do
    entity_element(car[i].e,free_cars,i,1);
```

The first line declares the list that holds cars not in active use. The remainder takes the entity part of each car, adds it to the tail of the list, sets its memnum to the car index number (i) and its typenum to 1.

Service Facilities

26. It is common for simulations to need certain facilities that are not essential for control purposes. For example random variates, statistics collection and presentation, and mathematical functions such as interpolation. A reasonable range is provided by SIMONE, but in part these services are based on facilities provided on the VAX system - specifically the uniform random numbers, trig functions in degrees, arc sin, arc cos, tan, a full 360° tan, log (base 10), and exponentiation (**).

27. Mathematical Functions. Table 1 gives details of the SIMONE functions and procedures provided in addition to normal PASCAL routines. Unless otherwise stated the results and parameters are of type REAL.

Table 1. SIMONE EXTENSIONS TO PASCAL MATHS FUNCTIONS

FUNCTION	COMMENTS
SGN(X)	0, +1 or -1 according to the sign of X.
RADS(X)	Converts X degrees to radians.
DEGS(X)	Converts X radians to degrees.
SINDEG(X)	sin (X degrees).
ARCSIN(X)	$\sin^{-1}$ (X) rads.
COSDEG(X)	cos (X degrees).
ARCCOS(X)	$\cos^{-1}$ (X) rads.
TAN(X)	tan (X rads).
TANDEG(X)	tan (X degrees).
FARCTAN(X,Y)	The arc tan of (Y/X), with quadrant selection based on the signs of X and Y.
LOG10(X)	

28. Cumulation. Coding in PASCAL can involve long concatenations of elements. For this reason it is convenient to have some simple procedures that increment and cumulate. Those available are listed in Table 2. The overhead in optimised VAX PASCAL for using these procedures is nil because the compiler takes single line subroutines to be 'in line subroutines', where the call is replaced by the body of coding.

Table 2. SIMONE EXTENSIONS TO PASCAL MATHS PROCEDURES

PROCEDURE	COMMENTS
INCR(I)	Integer parameter is incremented by 1.
INCRR(R)	Real parameter is incremented by 1.
CUM(I,J)	Integer parameters. I is increased by J.
CUMR(X,Y)	Real parameters. X is incremented by Y.

29. Interpolation. The three functions that are available for interpolation use linear interpolation, and produce real values (Table 3). The first is R2PT_INT which produces the Y value corresponding to an X value, given X and Y values on each side. The second is INTERPOLATE which produces the Y value corresponding to an X, given a vector of X values and an associated Y vector. Vector element indexes run from 0 to the constant TABLE_SIZE (currently 10). The vectors must be declared by the user as being of type RTABLE. The third is interpolate_1 which interpolates from an ordered list of any size. The list must be declared in the VAR section, of type data_list and declared in the coding with procedure declare_data_list. Data may then be added (in order, x increasing or x decreasing) using add_datum and add_dataX where X is 2-5. These procedures operate on a 'push-down' list where the first entry (x1,y1) is the furthest from the top of the list.

Table 3. INTERPOLATION AND RELATED SUBROUTINES

SUBROUTINE	PARAMETERS AND COMMENTS
R2PT_INT	Xvalue, Xlower, Xupper, Ylower, Yupper
INTERPOLATE	Xvalue, Xtable, Ytable
INTERPOLATE_L	listname, X value
DECLARE_DATA_LIST	listname, 'listname'
ADD_DATUM	listname, x,y
ADD_DATA2 -ADD_DATA5	listname, x1, y1, x2, y2, ... All four procedures use ADD_DATUM.

30. Random Variates. Most simulations use random numbers drawn from an appropriate distribution. When the distribution is not uniform they are called random variates. Mostly, random variates are obtained from the uniformly distributed random numbers using transformations, resampling schemes or by combining random numbers. If the uniform generator used does not produce numbers with suitable properties, this may well affect all the distributions from which the variates are actually drawn.

31. A pseudo-random number generator is generally provided in most computers. It is probably wise to check that the numbers produced do in fact have a reasonable range of properties. Many tests are available, some of which are in Knuth (Reference 4) and McMillan (Reference 5).

32., Uniform Random Numbers. The VAX version of SIMONE produces numbers through function RAND(X) where X is a seed. In general the seed value should not be assigned a value after the initial setting, since the function uses it to produce the next number. Exceptions to this general rule implemented in the function RANDOM allow sequence re-use and antithetic number generation. The starting seed should be a complicated number, since simple ones can produce abnormal serial correlations. Function GET_SEED produces a suitable starting seed given a simple starting seed. A different seed will be produced for each call to it so a number of separate streams can be set up. To ensure that results can be replicated care must be taken to set the variable SEED to some non-zero starting value and to initialise all streams before using any of them. In order to have some useful properties available to each stream a RANDOM_NUMBER_TYPE record has been defined. It allows sequences to be repeated, and antithetic sequences to be generated (the antithetic of U is $1 - U$ for $0 \leq U \leq 1$). A further property enables these options to be bypassed totally. Table 4 gives details of the record elements.

Table 4. RECORD RANDOM_NUMBER_TYPE

ELEMENT	COMMENTS
Current_seed	-
Starting_seq_seed	Current_seed is set to this value when the sequence is to be repeated.
Start_seq	When TRUE it is set to FALSE and starting_seq_seed is set to current seed.
Reuse_seq	When TRUE it is set to FALSE and the current seed re-set.
Quick	When TRUE no tests are done for sequences or antithetics.
Antithetic	When TRUE, antithetic numbers are produced if re-using a sequence.
Antithetic2	TRUE when antithetic is TRUE and a sequence is being re-used.

33. For each random number stream that is needed, a variable of the appropriate type should be declared. These variables can be initialised by calling procedure INIT_SEED for each one. A call to GET_SEED is included. To use the streaming, sequencing and antithetic facilities, function RANDOM should be used when generating uniform random numbers.

34. To re-use a sequence of numbers, the seed in use is set back to the value it had at the start of the sequence. This point must be defined by ensuring element START_SEQ is TRUE at that time. At the re-use time, element REUSE_SEQ must be set to TRUE. Production of antithetic variables will require both of these elements and element ANTITHETIC must be TRUE during the re-use. To bypass the tests for these factors QUICK should be TRUE. Typically the following might be seen in a program:

```
  var
    rn:random_number_type;  x:real;
    .
    .
    rn.start_seq:=true;  x:=random(rn);
    .
    .
    rn.reuse_seq:=true;  x:=random(rn);
```

35. Non-Uniform Random Variates. All of the distributions available are listed in Table 5 and have been tested using the VAX generator. Mostly the algorithms have been drawn from Knuth (Reference 4) although the Beta comes from Schmeiser and Babu (Reference 6) with corrections (McMillan) as well as from Knuth. They all produce a single real number unless otherwise stated, and require a parameter variable of type RANDOM_NUMBER_TYPE. Other parameters are real. The coding used for the Gamma distribution was chosen because the results accord with theoretical moments. Small changes in the higher moments can have disproportionate effects on the Beta distribution which sometimes calls the Gamma.

36. Statistics Collection. Many forms of data collection are used to obtain the required information from a model. One of the most common is the histogram, to which the simulation adds new observations from time to time. SIMONE provides procedures to set up the histogram, add values to it and to print it out along with its mean, variance and standard deviation (Table 6).

37. Initially the user must declare a variable for each histogram, of type HISTOG, and in procedure HISTOGRAM should declare its size (i.e. number of cells excluding the 'off the bottom' and 'off the top' buckets), its lowest collected value and the width of each cell.

38. Since the histogram is held as a linked list, there is no limit to the size of the histogram, and hence its accuracy. Non-linear cell size requirements need to be handled by the user, and transforms (e.g. log) on the values added to the histogram may be the answer. The procedures are sufficiently small and simple that specialised alterations should be practical.

Table 5. SUBROUTINES RELATED TO RANDOM VARIATE DISTRIBUTIONS

FUNCTION	PARAMETERS	COMMENTS
Poisson	Mean, RN	Gives an integer result.
Normal	Mean, SD, RN	
Exp_deviate	Mean, RN	Exponentially distributed variates.
Gamma	Mean, variance, RN	Gamma density $kx^{a-1}\exp(-x/b)$ (mean ab and variance ab^2).
Chi_squared	Mean, RN	Gamma (mean, 2* mean, RN).
Setupbeta (procedure)	p, q, C, D X, F, L A, PP	The specific parameters of the beta distribution must be known and a call to this procedure made to set up the required constants. The density is kx**(p-1).(1-x)**(q-1) and the variate is scaled as Cx+D. The other parameters must be real arrays of size 1..5 (X, F, L) or 1..10 (A, PP).
Beta	X, F, L, A, PP, RN	Parameters as defined above. The mean is Cp/(p+q)+D and the variance is $C^2.pq/(p+q)^2(p+q+1)$.
Sample	DISTX, DISTY, RN	Samples from a distribution defined in DISTX and DISTY, type RTABLE - array [0..TABLE_SIZE 10] of real. The X values must start at 0 and finish at 1. An interpolated Y value is produced.
Sample_1	dist_list,RN	Samples from a distribution defined using the ordered data lists described in the interpolation section. The X values must include 0 and 1.

Table 6. STATISTICS COLLECTION PROCEDURES

PROCEDURE	PARAMETERS AND TYPE	COMMENTS
HISTOGRAM		Sets up the necessary structures and initial values for a histogram.
	Table: HISTOG	Actual parameter must have been declared in the users VAR section.
	S: Real	Size of histogram excluding out of bounds buckets.
	l: Real	Lowest cell value.
	w: Real	Cell width.
ADDTO	Table: HISTOG Val: Real	Increments the count in the cell whose lower value is less than 'val' and whose upper value is not greater than 'val'.
WRITEHIST	F: Text Table: HISTOG T: String	Outputs to file F the title (T) and the contents of 'table' as well as its mean etc. Each cell takes one line.

List Handling

39. Apart from the demands of the SIMONE supervisory system for lists, simulations frequently have queues which can also be treated as lists. Relevant subroutines are listed in Table 7. In this implementation, apart from the histograms and data lists, only lists of entities are available, although extensions to other things should be possible.

40. Every queue or list must be declared as a variable of type LIST and must be set up using DECLARELIST. This procedure demands a name for the list so that associated error messages are significantly clearer.

41. The structure of a list is an information cell (identifying size and name, and a pointer to the head of the list), and list elements. Each list element points to both the next and previous elements in the list which is taken to be fully circular (a list of 1 will point to itself). Thus the 'previous' element to the first is the last element, and the 'next' element to the last is the first. In addition, each element points to an entity. Figure 4 illustrates this.

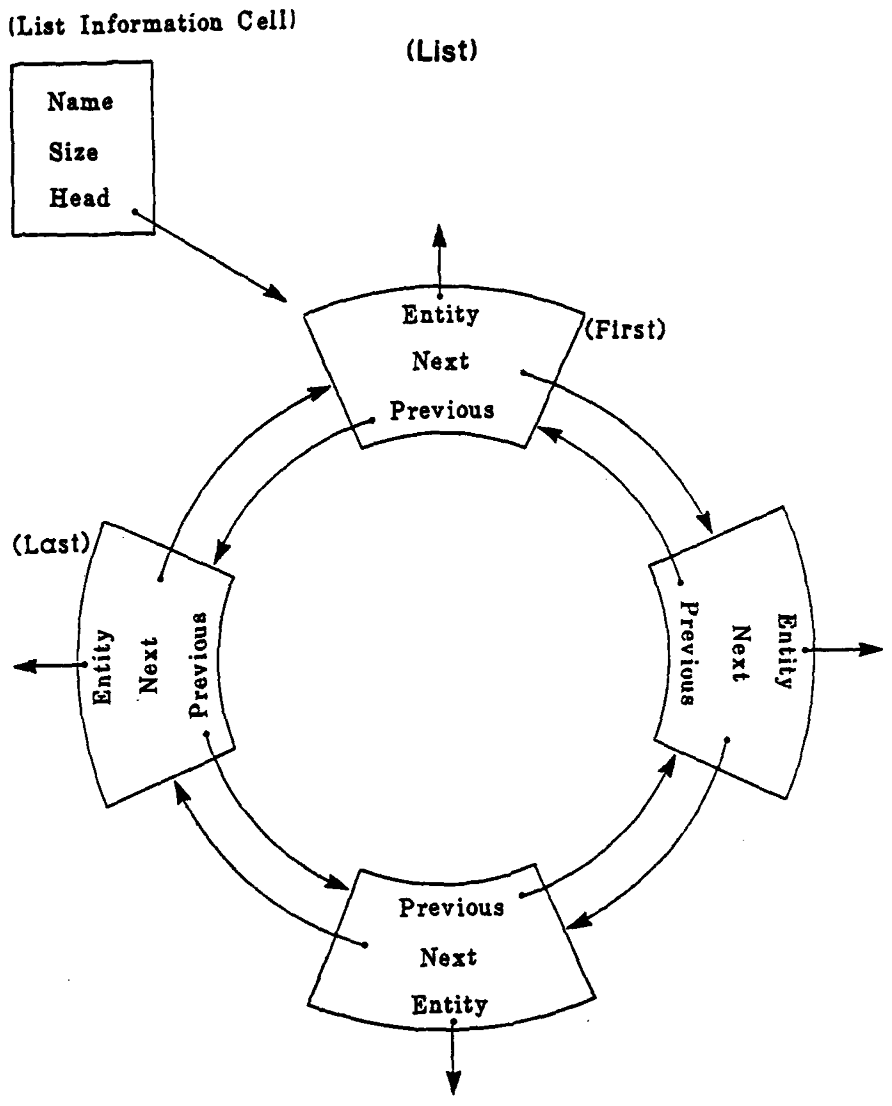


Figure 4. LIST STRUCTURE

42. Given this structure it becomes clear that one may rotate a list (in either direction) simply by changing the element that 'head' points to. Also, determining the head or tail of the list is quite straightforward.

43. Removing elements from a list is achieved through BEHEAD or BETAIL (which uses BEHEAD and a reverse rotation). On removal, links are suitably re-organised and the removed element returned to general computer storage with a 'dispose' instruction. The user may remove other elements by rotating, beheading and then restoring the original order with reverse rotation. If a particular entity needs to be removed from a list (rather than a list element) the procedure DELETE can be used. The list is rotated until the relevant item appears at the top, it is beheaded, and then rotated back to its original position.

44. Adding elements to the list is achieved through ADDLAST or ADDFIRST (which uses ADDLAST and a reverse rotation). Elements which are added must have an entity associated with them through one of the procedure parameters. Using the earlier example (Entity Attributes section) the cartype element e would be used.

Table 7. LIST SUBROUTINES

SUBROUTINE	TYPE	PARAMETERS	COMMENTS
DECLARELIST	Proc	LIST, name	
TAILOF	Func	LIST	Type list element pointer
ROTATE	Proc	LIST, number	number = -1 puts the tail at the head. Number can take any integer value.
BEHEAD	Proc	LIST	
BETAIL	Proc	LIST	
ADDLAST	Proc	M, LIST	M must point to an entity.
ADDFIRST	Proc	M, LIST	As ADDLAST.
VIEW	Proc	LIST	Used for debugging. Outputs a list name, size and element entities times, member and event numbers.

Supervisory Facilities

45. Control of time and event sequencing is achieved using a list called TIMEQ. Individual entities are added to this list through ADDTOTIMEQ which has parameters identifying the entity, the number of the event and the time of occurrence. For convenience, procedure ADDHDTOTIMEQ uses the head of a (nominated) list to identify the entity. Procedure SCAN looks through the times of the entities in TIMEQ for the smallest, stopping at the first one equal to CLOCKTIME, if any. The 'head' pointer of TIMEQ is set to point to the selected list element. Thus after SCAN is called, it is useful to set a variable of type entity_ptr to point to the relevant entity, thereby giving easy access to its information. For example, if such a variable is entp:

```
entp:=timeq.head^.item
```

then access to time, refnum and memnum are through the simple concatenations entp .time etc.

46. It should be noted that events for a single entity cannot be 'queued' because the next event number and its time are part of the entity data structure. If it is necessary to have an interleaving of events occurring to an object, then additional entity pointers should be set up in the record structure of the object. For example the cartype definition in the Entity Attributes system could have e as an array of entity_ptr. Each pointer would then reference its relevant event.

EXAMPLE

47. To illustrate the use of SIMONE a car servicing station will be simulated. Cars arrive, are serviced and then exit. If the station is busy, cars will queue and statistics on queuing time will be collected. The arrival rate for cars will be exponential with a mean of 10 for the first 300 and then 30. Service time will be gamma with a mean of eight and a standard deviation of four (variance = 16). The station will close at the first moment after 300 when it has no more work. The B-phase events are:

B1 : a car arrives

B2 : a car exits

B3 : arrival rate reduces

There are two C-phase events:

C1 : to service a car

C2 : to stop the simulation when the last car has been serviced after time 300.

48. The flow diagrams for B1 to B3 and C1 to C2 are in Figures 2, 5 to 8 respectively. The object-server interaction diagram of Figure 3 also describes the situation.

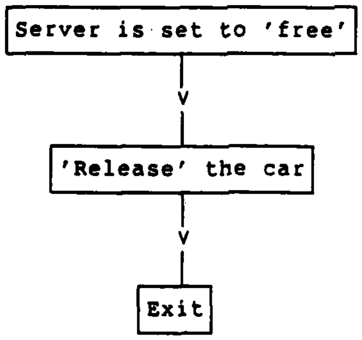


Figure 5. EVENT B2: CAR EXITS FLOW DIAGRAM

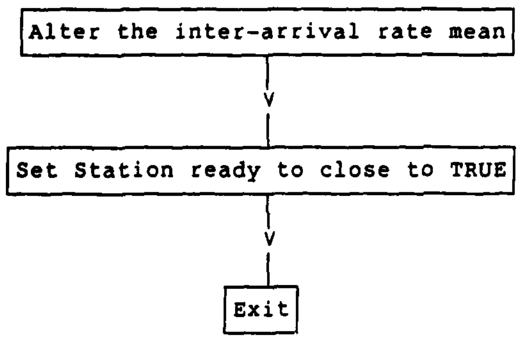

Alter the inter-arrival rate mean
Set Station ready to close to TRUE
Exit

Figure 6. EVENT B3: INTER-ARRIVAL RATE REDUCES

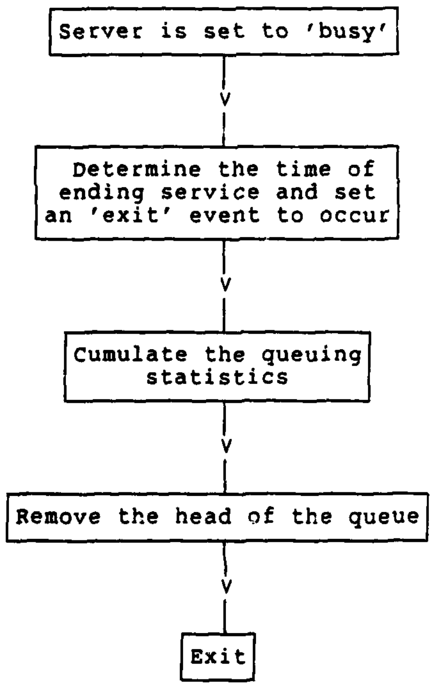

Server is set to 'busy'
Determine the time of ending service and set an 'exit' event to occur
Cumulate the queuing statistics
Remove the head of the queue
Exit

Figure 7. EVENT C1: SERVICE FLOW DIAGRAM

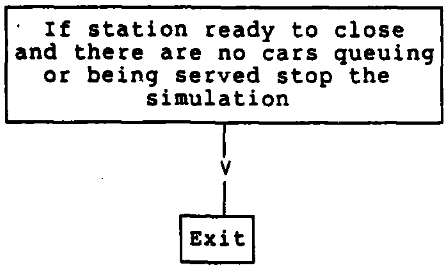

If station ready to close and there are no cars queuing or being served stop the simulation
Exit

Figure 8. EVENT C2: STOP THE SIMULATION

49. To reduce mistakes and enhance debugging, the CONST section may be used for key numbers. In particular, B events which must be numbered (since REFNUM is an integer) may be given a name in CONST. An alternative to this is through ORD and an enumerated list. In this example, the beginning of the program is as follows:

```
[INHERIT('SIMONE.PEN')]
program service_cars;
CONST
  arrive=1;
  exit=2;
  reduce_rate=3;
  max_number_of_cars=10;
```

The first line allows access to the precompiled SIMONE subroutines and the remainder is self explanatory.

50. The next section of program defines the types of variable in use, and is where any enumerated lists would appear. In this example, the only entity that is active is the car(s). While the server is an active part of the problem (Figure 3) and could be an entity, its involvement has become secondary because no statistics about it are being collected. The server has thus become passive and does not need entity status. The next part of the program is therefore:

```
TYPE
  cartype=record
    e:entity_ptr;
    join_q_time:real;
  end;
```

The link to SIMONE is through e, and apart from an index number, the only property that cars will need is the time that they joined the queue. Definition of the variables must follow, and should include all histograms, queues, random number streams, etc. Variables already declared by SIMONE are SEED, LEASTTIME, CLOCKTIME, ERROR and TIMEQ. The next part of the program is:

```
VAR
  car:array[1..max_number_of_cars] of cartype;
  rate_change:entity_ptr;
  server_busy, simulation_running, ready_to_close:boolean;
  free_cars, queue:list;
  mean_inter_arrival_time:real;
  rn:random_number_type;
  queueing_times:histog;
  next_item:entity_ptr;
  i:integer;
```

51. Most of these variables are self explanatory, but of note are 'rate_change' which is the entity that allows event B3 to occur, 'free_cars' which is a list of cars not actively involved (i.e. not queuing or being serviced), 'rn' which is the random number stream and 'next_item' which is a variable of convenience mentioned in the Supervisory Facilities section (as entp). Variable 'car' has been given a size of 10 (max_number_of_cars) simply because it is not expected that more than that number will be active in the simulation at any one time. Cars which have been serviced will re-enter the free cars list and may be re-used as necessary. Statistics on waiting times will be collected as they arise, in 'queuing_times'.

52. Definitions having been made, initialisation follows:

```
Begin
  {Initialisation section}                         {line number}
   seed:=12345;                                         {1}
   Init_seed (rn);                                      {2}
   Declarelist (timeq,'timeq');                         {3}
   Declarelist (free_cars,'free_cars');                 {4}
   Declarelist (queue,'queue');                         {5}
   Histogram (queuing_times,10,0,4);                    {6}
   For i:=1 to max_number_of_cars do                    {7}
     entity_element (car[i].e,free_cars,i,1);           {8}
   Addhdtotimeq (free_cars,arrive,0);                   {9}
   Behead (free_cars);                                  {10}
   New(rate_change);                                    {11}
   Addtotimeq (rate_change,reduce_rate,300);            {12}
   Server_busy:=false;
   Ready_to_close:=false;
   Mean_inter_arrival_time:=10;
   Clocktime:=0;
   Simulation_running:=true;
  {end initialisation section}
```

Lines 1 and 2 define the starting point of the random numbers and set up the stream to be used. Lines 3 to 5 set up all lists and queues to be used, particularly including the TIMEQ. Line 6 sets up the histogram with 10 cells (plus out of range buckets) starting at 0 and having a width of 4. Lines 7 and 8 declare each element of the car array to be a single entity and put it into the free_cars list. The entity index (memnum) is the same as its array index (i) and its type number (typenum) is 1. The type number is irrelevant in this example. Lines 9 and 10 take the first car from the free_cars list and put it into the timeq for the 'arrival' B event (B1) which will occur at time 0. Lines 11 and 12 sets the 'reduce rate' variable and set its event (B3) to occur at 300. The remaining lines need no further explanation.

53. The next section of code is the A-phase:

```
repeat{.until simulation running is false}
  scan(timeq);
  next_item:=timeq.head^.item;
  lasttime:=clocktime;
  clocktime:=next_item^.time;
  repeat
    behead(timeq);
    case next_item^.refnum of
      .
      .
      {B-phase events}
      .
      .
  end{case};
  scan(timeq);
  next_item:=timeq.head^.item;
until next_item^.time <> clocktime;
```

It should be noticed that the phase A code surrounds phase B. This is necessary so that all events occurring at any one time happen before entry is made to the phase C. Setting lasttime to clocktime before advancing time allows elapsed time to be calculated if needed, thus immediately before 'repeat' is the point at which any continuous variables in the model should be updated.

54. The phase B events follow, and are clearly identified by using the appropriate constant as the 'case' label and indentation.

```
arrive:
          begin
            addlast(next_item,queue);
            car[next_item^.memnum].join_q_time:=clocktime;
            addhdtotimeq(freecars,arrive,clocktime+
              exp_deviate(mean_inter_arrival_time,rn);
              behead(freecars);
          end;

exit:
          begin
            server_busy:=false;
            addfirst(next_item,freecars);
          end;

reduce_rate:
          begin
            mean_inter_arrival_time:=30;
            ready_to_close:=true;
          end;
```

55. Finally the C-phase events coding and end of simulation output is as follows:

```
{C1}      If (server_busy=false) and (queue.size>0) then


  begin server_busy:=true
    addhdtotimeq(queue,exit,gamma(8,4,rn)+clocktime);
    addto(queuing_times,
    clocktime-car[queue.head^.item^.memnum].join_q_time);
    behead(queue)
  end;

{C2}      If (ready_to_close) and (queue.size=0) then

  simulation_running:=false

until simulation_running=false;

  writehist(queuing_times);
  end.
```

56. There are other ways to achieve the same result, but this illustrates the general structure and requirements that most simulations will follow. Of particular importance are the need to:

a. schedule new events;

b. have an entity associated with that event;

c. add clocktime to the interval to the next occurrence of an event;

d. 'behead' queues (including the timeq) when the element is no longer needed; and

e. ensure that the phase A structure allows the occurrence of all phase B events at any one time before entering phase C.

SUMMARY

57. The general discrete event simulation structure required by the SIMONE supervisory system has been described. It requires phase A to control time, phase B to handle events that occur following the passage of time and phase C to handle events that occur as a result of conditions being met. Some advantages of using this approach have also been mentioned, that is its portability, flexibility, ease of use and structured programming approach. Principle differences between discrete event and continuous modelling processes and available functions have been canvassed, and the point within phase A at which continuous elements in a principally discrete event model should be inserted has been identified.

58. General facilities provided by SIMONE have been detailed. They include entity and list or queue handling, statistics collection and display plus random number and random variate production, as well as extensions to the standard Maths functions provided by PASCAL.

59. The formal flow diagram of SIMONE has been given, and the ways in which conventional flow diagramming fit within this structure illustrated.

60. Finally a very simple example has been used to show how the system can be used in practice.

GLOSSARY

Antithetic	Given a value with cumulative probability of occurrence p, its antithetic is that value associated with the cumulative probability of occurrence 1-p.
Composite Entity	Group entity or multiple group entity.
Continuous Models	Time progresses evenly and events play little spectific part.
Cumulation	Creation of a running total.
Discrete Event Models	Time progresses by convenient leaps (often of different sizes) on an event by event basis.
Entity	An object or thing to which events occur.
Events	Instantaneous happenings that often mark the start or finish of processes, when entity attributes change.
Group-Entity	A collection of entities of identical type and properties.
Interpolation	Knowing values for two locations, estimating the value at a third location between the first two.
Linear Interpolation	Interpolation assuming a straight line between the known values.
List	An ordered sequence of things.
Multiple Group-Entity (MGE)	A group-entity that can have several different B events occur to it.
Phase A	Controls time advancement and event selection.
Phase B	Contains time dependent events.
Phase C	Contains conditional events.

Pointer	The value in a PASCAL variable of this type refers to a memory location.
Queue	Almost synonymous with list. Generally applied when the 'things' are entities.
Random Number	A real number in the range 0 to 1 having all values equally likely.
Random Variate	A randomly chosen value with a probability of occurrence that conforms to a selected distribution.
Seed	A value needed by a pseudo random number (prn) generator to create a prn.
Supervisor	A controlling authority.
Timeq	A list of forthcoming events as implemented in SIMONE.

REFERENCES

1. P.R. Hills, 'SIMON - A Computer Simulation Language in Algol', Bristol College of Science and Technology Report No. MDM 1401, 1964.

2. P.R. Hills, 'SIMON', Paper presented to NATO Conference on Simulation, Hamburg.

3. P.R. Hills, 'An Outline of the SIMON Simulation System', Imperial College of Science and Technology, Production Engineering and Management Studies Section Report No. 66/4, 1966.

4. D.E. Knuth, 'The Art of Computer Programming. Volume 2 - Seminumerical Algorithms'.

5. B.K. McMillan, 'Some Tests for Uniform Random Number Generators', CSE Working Paper RESEARCH 5, 1982.

6. B.W. Schmeiser and A.J.G. Babu, 'Beta Variate Generation via Exponential Majorising Functions', Operations Research, August 1980.

7. B.K. McMillan, Program 'SIMONE1.PAS', Central Studies Branch, FDA Division, Department of Defence, Canberra, 1988.

DISTRIBUTION

	Copy No.
DEPARTMENT OF DEFENCE	
Central Office	
Deputy Secretary B	1
First Assistant Secretary, Force Development and Analysis Division	2
Headquarters Australian Defence Force	
Vice Chief of the Defence Force	3
Assistant Chief of the Defence Force (Policy)	4
Navy Office	
Navy Scientific Adviser	5
Director Operational Analysis - Navy	
Army Office	
Scientific Adviser - Army	6
Director Operational Analysis - Army	
Air Force Office	
Air Force Scientific Adviser	7
Director Operational Analysis - Air Force	
Defence Science and Technology Organisation	
Chief Defence Scientist	8
Assistant Chief Defence Scientist - Policy	9
Central Studies Branch	
Assistant Secretary, Central Studies	10
Principal Research Scientists, CSB (3)	11-13
Senior Service Representatives, CSB (3)	14-16
Author	17-27
Mr G. Sterns	28
Dr R. Smith	29
Dr A. Joye	30
Mr D. Bennet	31
Mr W. Barnes	32
Dr R. Wood	33
Mr W. Warton	34
Mr S. Brown	35
External	
Seng Kuan Tan, Malaysian Ministry of Defence	36
Technical Reports Centre DCL-CP (18)	37-55

Department of Defence

DOCUMENT CONTROL DATA

AD A195600

1. a. AR No AR-005-255	1. b. Establishment No CSB MEMORANDUM 2	2. Document Date March 1988	3. Task No
4. Title SIMONE, A DISCRETE EVENT SIMULATION SUPERVISOR		5. Security a. document U/C b. title U/C c. abstract U/C	6. No Pages 7. No Refs
8. Author(s) B.K. McMILLAN		9. Downgrading Instructions -	
10. Corporate Author and Address Central Studies Branch, Department of Defence, CANBERRA, A.C.T. 2600. AUSTRALIA.		11. Authority (as appropriate) a.Sponsor b.Security c.Downgrading d.Approval a-d CSB	

12. Secondary Distribution (*of this document*)

APPROVED FOR PUBLIC RELEASE

Overseas enquirers outside stated limitations should be referred through ASDIS, Defence Information Services Branch, Department of Defence, Campbell Park, CANBERRA ACT 2601

13. a. This document may be ANNOUNCED in catalogues and awareness services available to ...

NO LIMITATIONS

13. b. Citation for other purposes (*ie casual announcement*) may be (*select*) unrestricted (*or*) as for 13 a.

14. Descriptors Simulation Languages Simulation Monte Carlo Method	15. COSATI Group 0072E

16. Abstract

The general discrete event simulation supervisory system SIMONE is described. The facilities available and method of use are detailed and illustrated with an example. It uses a three phase system to control time and event selection, and provides additional mathematical, random sampling and data collection facilities. Systems with a PASCAL compiler should be able to use SIMONE.

END

DATE
FILMED

8 8